PATENT APPLICATION
Docket No.: 13768.783.108

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of		)
	Pierre-Jean Reissman, et al.	)
Serial No.:	10/677,129	) Art Unit) 2192
Conf No.:	3890	)
Filed:	September 30, 2003	)
For:	METHOD AND SYSTEM FOR AUTOMATICALLY TESTING A SOFTWARE BUILD	)
Examiner:	Thuy Chan Dao	)

DECLARATION UNDER 37 C.F.R. § 1.131

I, Jacqueline L. Reid, declare as follows:

1. I am a co-inventor of the invention claimed in the above-identified patent application. I am employed by Microsoft Corporation, the assignee of the above-identified patent application.

2. During my employment at Microsoft Corporation, and prior to July 31, 2003, I (along with my co-inventors) conceived and reduced to practice in the United States the Invention as described and claimed in the above-identified patent application.

3. Attached as Exhibit A is a copy of a document entitled "AScan: binaries content, behavior and dependencies scanner" last modified on March 3, 2003. (hereinafter referred to as "AScan"). Ascan includes a description of the invention including a number of Figures that were

included in the present application. Ascan further includes source code on pages 6 and 7 dated February 17, 2003 showing an actual reduction to practice of the invention. Page 8 illustrates results from a use of the invention.

4. I was personally involved with the conception and reduction to practice of the concepts described in Ascan.

5. I declare further that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful, false statements may jeopardize the validity of the application or any patent issuing thereon.

Dated this 29th day of October 2007.

Inventor: Jacqueline L. Reid

EXHIBIT A

AScan: binaries content, behavior and dependencies scanner.

Goal:

The Adaptive Scanner (AScan) is a tool that will automate the tracking of changes between builds. The tool will provide:

- Full-Automatic basic invocation of the binaries (defaults parameters)
- Flexible configuration for smarter validation (smoke test / validation test/ Activation Models)
- Selective testing of classes and dependent classes.
- Fault injection to maximize coverage
- Tracking and reporting services.

Each class/method and their dependencies will be scanned and memorized in the system.

AScan memorizes both the content and the behavior of the binaries in a detailed dictionary. This dictionary will be automatically updated with the deltas between builds. AScan will use the deltas to create reports and tasks to quickly handle the changes in the new build and assess its quality.

- The binaries content is based on the MSIL of the binaries; it contains all the members of the types as well as the dependencies on other types.
- The binaries behavior is captured by directly invoking the methods with sets of parameters. The results of the runs are memorized and any change in the method signature/code/results will be reported. A report of impacts on the dependents (code dependency) will be presented to the user. The item of the report may be used as tasks for test work on the new build. Tasks may be filtered to target only specific areas of the product.

The tool will allow adding external providers to perform dedicated testing of the classes, like providing a constructor or a data provider for exotic types. AScan will allow using instrumented builds to report on class/code coverage. By default, AScan will perform its tests on all classes independently of their accessors (public/protected/private/internal).
The tool will send selected notifications to its users about events that occur during testing. For instance, a tester may be interested in the "all new types" and "behavioral changes for X and Y types" events.
The tool will provide great flexibility in configuring the tests and reporting results by allowing users to access a very high level of granularity for the configuration of tests and the analysis of results.
AScan will integrate seamlessly with the production system to create bugs, track and follow regression, assist users in handling the tasks of their workflow.

Non Goals

AScan does not automatically test high-level integration scenarios. AScan does not provide ad hoc tests. AScan does not do your testing job.
AScan does not replace our test infrastructure, it will integrate with the infrastructure.
AScan is not the unifying repository for the project definition. This repository is the global graph (HUB) of the projects that links requirements, scenarios, specifications, designs docs and deliverable to the workflows <spec TBD>.

Scenarios:

Tina Foo is a test lead in charge of a feature of the product. Builds are dropping daily and code churn is just driving everybody in the team crazy. AScan changed their live by automatically and quickly providing a GPS to navigate the 300+ classes of the assemblies. Tina uses AScan to dispatch the new areas to her team and correlate the binaries to be tested with the spec and the design.

Joe Bar is a tester in charge of 3 specifics features. His features have many dependencies on the core .dll and other lateral providers. Managing changes is essential in his daily work. AScan handles the deltas for him in a human friendly way. Accessing the results and exploring the changes is as simple as a click. Communicating with his peers and colleagues from the dev team has never been so easy. Tracking regression results is seamless. AScan provides the history of behavioral changes as well as the static changes of the code.

Sue Perr is a SDE/T providing some infrastructure code to help her buddies testing by creating smart factories. She is delighted to turn her factories in data providers for AScan. Her work is now shared across the whole team and AScan is just another special consumer.

Boba Fett is a developer in GDI+, running prechecks prior to check in and understanding the impact factor of his changes on the product positively changed his days. Communication with test is easier since data providers, repro code, results and history are seamlessly available to both parties.

Mike O'Phone is a test manager and he is interested in getting the pulse of the products. The reporting services of AScan give him all the statistics he's been looking for. Each delta of a new build is split into categories as Creation, update or Deletion. The dynamics of the system are captured as well. Stability starts showing in the product since both static and dynamic behavior have turn to green light. Less and less evolutions of the architecture are reported and code tends to repro results between builds.

Architecture

AScan has a 3 tiers architecture with a rich client. The client embeds the User Interface to configure and run local testing as well as configuring global testing on the server. A second purpose of the client is to analyze the results of the execution of the tests. The server has mainly storage querying and analysis purposes. The server stores the global configuration of the tests, the results of the execution for multiple configurations, tasks and user specific information regarding testing.
The Test engine is available in the client for local testing. The same engine is running bi-daily on various configurations to report the global state of the build.
A first pass is executed right after the new drop of the build around 4 am, Testers work on the configuration to integrate the delta and new results are uploaded the same day around 2 p.m.

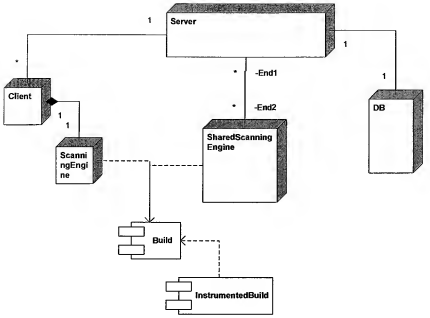

Figure 1. AScan architecture

Execution Modes

The architecture supports online and offline modes. Configuration and results are serialized using XML schemas. Switching between online and offline modes is almost transparent to the user. The application may be used in one of the following mode:

Offline: the user works on her machine, outputs are stored locally, a cached configuration is used. Offline is also used to work on the delta and prepare the add-in to the global configuration.

Online: the user is connected to the server and the rich global dictionary is available.

Dictionary

The Dictionary is built on the core architecture. The core architecture is a tree with the following structure:

{ Product {Build {Assembly {Type} } } }

An initial dictionary is produced the first time a product is processed with AScan. Each build of the product is scanned and memorized as a new version of the dictionary.

The delta between the builds is automatically generated by AScan and contains three trees {Deleted, Unchanged, Created}. These trees contain all the members of the types as well as their dependencies. All the alterations generate events <Created, Updated, Deleted>. The Unchanged and Created trees also contain the dynamic information about the execution of the code. If the output is different from the previous run, an <Behavior> event is generated by the method.

Core architecture

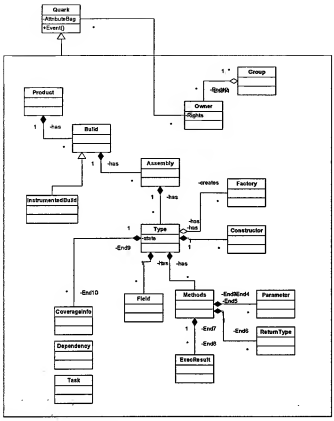

Figure 2. Core Static Architecture

The main object in the memory of the scanner is a build. AScan is essentially a diff tool to capture the delta between builds. A build is a collection of Types. These Types contain both Fields (Data) and Methods (Operating Code). Any entity may have dependencies (Dependency on another assembly for an assembly, Type usage as a field or a parameter for a Type, method call in the body of a method of a type). Methods are invoked with default or ad-hoc user specified parameters and the result are stored. Subsequent invocations shall produce the same results or a behavioral difference has been spotted out. Methods are invoked on automatically constructed objects using the constructors of the class or using the factories provided by the user. The expected behavior is an ExecResult, it might be specified by the user. If no ExecResult is specified, AScan uses the returned result it got from the invocation as the expected result.

Access mask:

All classes of the core derive from the Quark class. The Quark Class provides basic ACLs and ownership on the objects of the system. Quarks have a binary access mask used to derive Read/Write/eXecute (1-2-4) rules. For instance 111 – 101 – 100 (751) allows full access to the owner, read only and execute access to the group and read access to the rest of the world.

Mask	R-W-X	R-W-X	R-W-X
Domain	User	Group	World

Figure 3. Access mask of the Quarks.

Scanning engine

The scanning engine (SE) is a multithreaded agent responsible for the execution of the tests. It is robust to support resiliency to crashes and other failures of the binaries under test. It produces an XML stream that will be consumed by the client and the server depending on the mode of execution. SE communicates with a client (the AScan client, the AScan server or any other process) to receive the configurations to be invoked. SE executes and reports results.

The SE implements the IScanEngine Interface:
<TBD>

The following workflow is used to produce a dictionary:

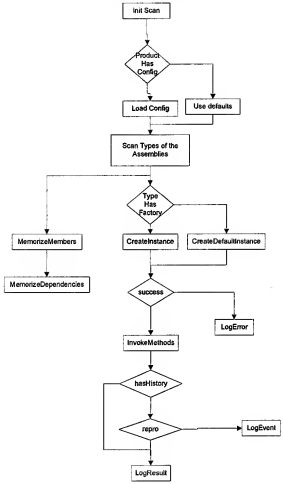

Figure 4. Dictionary generation workflow

SE is invoked to scan assemblies. User may add pre process, factories, vector of parameters, comparators and post process to a specific type/method. These add-ins are called article of configuration, together they form the configuration of a dictionary. If an existing configuration for a dictionary exists, the product is passed to SE. SE uses the configuration to initialize the tests. If no configuration has been defined, SE will use default parameters. A configuration may be partial, sometimes a Type does not need to receive much attention and default parameters will be sufficient. Some articles of configuration will not apply anymore because the binaries have changed and the method or type may have been altered or deleted.

When an article of configuration does not apply anymore it is flagged with a <misconfiguration event>. It is up to the user to archive it or discard the component.

Once the configuration is loaded, the assembly is scanned and stored in the tree of the new dictionary. Each type is scanned: members and dependencies are stored, if selected by the configuration, each method is invoked and a result is logged. If the result is not the expected result, a "<behavioral event>" is logged.

```
<SmokeTest date="2/17/2003 6:10:57 PM">
  <Assembly name="SmokeIt.exe">
    <Class name="ConsoleX" namespace="SmokeIt">

      <method name="Finalize" public="False" static="False" returns="System.Void" param="" />

      <method name="GetHashCode" public="True" static="False" returns="System.Int32" param="" />

      <method name="GetStdHandle" public="False" static="True" returns="System.Int32" param="System.Int32
hStdHandle,">
        <param val="(System.Int32 'null')" />
        <ret value="-1" mode="static nullParam" />
        <param val="(System.Int32 '0')" />
        <ret value="-1" mode="static param" />
      </method>

      <method name="GetConsoleScreenBufferInfo" public="False" static="True" returns="System.Int32"
param="System.Int32 hConsole, SmokeIt.CONSOLE_SCREEN_BUFFER_INFO& cbInfo,">
          <param val="(System.Int32 'null') (SmokeIt.CONSOLE_SCREEN_BUFFER_INFO& 'null')" />
          <ret value="0" mode="static nullParam" />
          <param val="(System.Int32 '0') (SmokeIt.CONSOLE_SCREEN_BUFFER_INFO& 'null')" />
          <ret value="0" mode="static param" />
        </method>

      <method name="WriteConsole" public="False" static="True" returns="System.Int32" param="System.Int32
hConsole, System.String pText, System.Int32 nTextLength, System.Int32 nWritten, System.Int32 nReserved,">
          <param val="(System.Int32 'null') (System.String 'null') (System.Int32 'null') (System.Int32 'null')
(System.Int32 'null')" />
            <ret value="0" mode="static nullParam" />
          <param val="(System.Int32 '0') (System.String
'tŸôÛÚžáçQÇ@xš%[Këë9%ZM/ÿ4ÓUÁÔ2ývHm{ÊÃñÐðÍb-
T0zjuy·RDGÁŸòÛíá*1?ïmGõeÒú(G8F8ºXJkewl\û-Ã]Ú6lgãÕùQívlÚ_JÅ{àVj') (System.Int32 '0') (System.Int32 '0')
(System.Int32 '0')" />
            <ret value="0" mode="static param" />
      </method>

      <method name="WriteX" public="True" static="True" returns="System.Byte" param="System.Object[] olist,">
          <param val="(System.Object[] 'null')" />
          <err value="System.NullReferenceException: Object reference not set to an instance of an object. at
SmokeIt.ConsoleX.WriteX(Object[] olist) in C:\TMPDSK\Dev\exp\exec\SmokeIt\ConsoleX.cs:line 81"
mode="static nullParam" />
          <param val="(System.Object[] 'null')" />
          <err value="System.NullReferenceException: Object reference not set to an instance of an object. at
SmokeIt.ConsoleX.WriteX(Object[] olist) in C:\TMPDSK\Dev\exp\exec\SmokeIt\ConsoleX.cs:line 81"
mode="static param" />
      </method>
```

Figure 5. Sample of a dictionary

Note: the non static method could not be invoked since it was not possible to automatically create an instance of the class ControleX. On the other hand static methods were successfully invoked.

```
<SmokeTest date="2/17/2003 6:10:57 PM">
  <Assembly name="SmokeIt.exe">
    <Class name="ConsoleX" namespace="SmokeIt">
      <method name="GetConsoleScreenBufferInfo" public="False" static="True" ... >
        <param val="(System.Int32 'null') (SmokeIt.CONSOLE_SCREEN_BUFFER_INFO& 'null')" />
          <ret value="0" mode="static nullParam" />
```

Figure 6. Sample of a method invoke result showing the Core Tree structure

Delta Generation

Once the new build has been scanned by the ES, a delta with the previous build (if it exists) is generated.

Note: Deltas may be computed between any versions of the Dictionaries.

Each element within the tree of a dictionary is uniquely identified by its path

`(((namespace).type).field)`

or

`(((namespace).type).method.signature)`

Each element will be flagged as Created, Unchanged or deleted. (<OLD> and <NEW> in the sample). Let's work on a sample to clarify the ideas.

The following sample has been generated on the binaries of a feasibility prototype of the present spec. The prototype contains several classes with multiple methods and fields. The current prototype processes only methods and behavioral differences. Fields and dependencies may be included without theoretical or practical issue.

The <OLD> (resp <NEW>) tag is used to prefix any entry of the previous (resp current) build dictionary that was deleted (resp added) in the new build.

```
<OLD> <RegExec> SmokeIt:AScan:True.False.GetHashCode()::RETobject param:115
<OLD> <ExcExec> SmokeIt.bar.True.False.doit()::ERRoject nullParam:System.Exception: Did AScan catch the behavioral diff?
  at SmokeIt.bar.doit() in c:\tmpdsk\dev\exp\exec\smokeit\smokeit.cs:line 28
<OLD> <ExcExec> SmokeIt:bar:True.False.doit()::ERRobject param:System.Exception: Did AScan catch the behavioral diff?
  at SmokeIt.bar.doit() in c:\tmpdsk\dev\exp\exec\smokeit\smokeit.cs:line 28
<OLD> <RegExec> SmokeIt:AScan:True.False.GetHashCode()::RETOject nullParam:115



<NEW> <RegExec> SmokeIt:bar:True.False.doit()::RETOject nullParam:null
<NEW> <RegExec> SmokeIt:bar:True.False.doit()::REFobject param:null
<NEW> <RegExec> SmokeIt:AScan:True.False.GetHashCode()::RETobject param:116
<NEW> <RegExec> SmokeIt:AScan:True.False.GetHashCode()::RETOject nullParam:116
```

Figure 7. Sample of a Delta.

The method doIt() from the class SmokeIt.bar used to throw an exception when invoked. The present build has either fixed the problem or created a new one, anyway AScan has detected the behavioral difference and reported it. Now it is up to the user to accept the new behavior or check with the developer (and the PM) the reason for the change and decide if it is a bug or not.

The method GetHashCode() inheritated from System.Object is returning a different value: 116 instead of 115. For a variation of the returned value, the user could either take the event as a behavioral difference or define the expected return value as being of type int only. To be more radical the user might just decide to ignore these inherited methods in her report of the delta.

Code coverage

The following coverage reports were generated from an instrumented version of System.Messaging.dll:

1) A direct scan (no configuration – full automatic mode)
2) A simplistic configuration where some additional Types instanciators (10+ out of 100+ required, 2 minutes of manual work that could be partially automated) where integrated to a Factory to provide initialized parameters to the invocation of methods.

1) direct scan

All Components	Blocks	Arcs	Functions	Source Files	Classes
Total	6411	7100	796	42	58
Hit	2907	2612	713	40	53
Percentage (%)	45%	37%	90%	95%	91%

System.messaging.dll	Blocks	Arcs	Functions	Source Files	Classes
Total	6411	7100	796	42	58
Hit	2907	2612	713	40	53
Percentage (%)	45%	37%	90%	95%	91%

2) simplistic configuration

All Components	Blocks	Arcs	Functions	Source Files	Classes
Total	6411	7100	796	42	58
Hit	3158	2925	719	40	55
Percentage (%)	49%	41%	90%	95%	95%

System.messaging.dll	Blocks	Arcs	Functions	Source Files	Classes
Total	6411	7100	796	42	58
Hit	3158	2925	719	40	55
Percentage (%)	49%	41%	90%	95%	95%

The coverage improvement for blocks and arcs between the 2 runs is 4% for both buckets.
The whole process took less than 5 minutes.

GTO and the stress engine are ideal candidates for providing advanced sub engines for both the parameter validation and the type factory.

Reporting services

It is straightforward to extract code build churn statistics, code coverage and behavioral mutations from the new dictionary.

- **Code build churn:** A report shows all new/unchanged/deleted types. The user may drill down to the finest level of details. Methods contain the number of line in MSIL. This is a good indicator (flattens dev hyper-super convoluted code) of the code structure.
- **Code coverage:** A must to have, coming for free thanks to Magellan
- **Behavioral mutation**: DEV scenario: "I check in a fix, everything runs smooth". AScan "oops" it runs fine on your specific scenario but your fix introduced sneaky side-effects.

Echelon or the dependency graph will point out the impact of any modification and run the specific minimal CI tests on the impacted set of types.

- **Code generation:** In case the user needs an independent rerun source code, AScan will generate the source for different targets (component,assembly,exe) / different languages.

User Workflow

Once the delta has been generated users may start looking at the differences and start working on the new build. For statistics and traceability, users are authenticated and have specific permissions: {Read, Write, Exec, {Admin} }.

Note user refers indifferently to a user or a group of users. See the access mask section for details.

- Admin: an admin has Full access anywhere In the System.
- Read: a Read access is granting read access to the Quark
- Write: a Write access is granting write access to the Quark
- Execute: an Execute access is granting execution of the Quark (Exec must be supported by the Quark)

The group structure reflects the feature teams and allows local partitioning of the namespaces.

Based on their rights users may perform the following tasks:

- Admin-Users have to dispatch the new Types to users. By default new types may be assigned by users themselves and reassigned by the admins. Once the type has been assigned only the owner or an admin may reassign the type.
- Write-Users may start tuning the configuration for their types locally and upload the updates to the AScan repository.
- Execute-Users may run the configuration to produce results, refine their validation until they are ready to check in their work. Rerun some scan to repro and debug.
- Read-Users may start exploring the delta information and fill in bugs in case of discrepancies.

User Interface

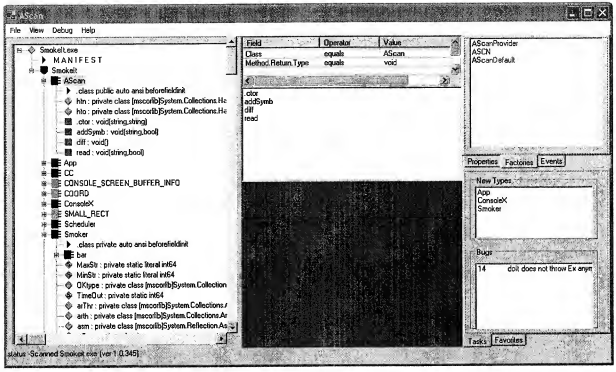

Figure 8. MDI Interface for AScan

Component	Comment
MDI	The client is a MDI application – VS like
MenuBar	The Menu Bar has the following items: File, View, Debug and Help
TreeView	The dictionary explorer. Allows to access any object in the tree. Context menu to do most common operations
Query Builder	Another way to access information. Clauses are built on the metadata of the Core hierarchy
Result list	Result from the query
Tab control (Dictionary)	Contains Dictionary specifics: Property. Factories and Events
Tab Control (user)	Contains User specifics: Task and Favorites.

Features TBD:

Build process
Code coverage
Dependency manager
Settings
Rerun
Reports
Admin – work assignment
Bug track integration
Code spit
Configuration – enable disable / run (local) global

adaptive scaner
scan binaries
extract dependencies
extract blue print - APIs
UI tree navigation - registers plugins/ helpers
provide defaults - autoscan (pass default params to methods)

memorizes the behavior reports on diffs (behavior changed berween builds)
memorizes the footprint - provides smart diff with notification -> workflow (lead dispatches new items to the team)

tightly coupled with code coverage -> reports, indicator, trends,stats, Where to concventrate ...
bugs tracking

global vision of the product

HUB / market place for testing the prod

reporting/analyzing/querying tools

online/offilne -> XML/SQL transparent

coupled with the TCM/Automation

reuse - factorize init cleanup in the helper
AClass -> AClassDrv

Stress engine / Factories : provides consrtructors (init) - destrucotrs (cleanup)
TCM integration – AScan invoke bi-daily
GTO / TMT / ITE / Pict – provide combinatorics - validation
GTO integration
Fault injection / parse MSIL – mute and run

```
IL_0085:  newobj     instance void mscorlib]System.ArgumentException::.ctor(string)
```